Haskell

The Eq Class

```
(==) :: Eq a => a -> a -> Bool
(/=) :: Eq a => a -> a -> Bool
```

The Ord Class

```
(<) :: Ord a => a -> a -> Bool
(>) :: Ord a => a -> a -> Bool
(<=) :: Ord a => a -> a -> Bool
(>=) :: Ord a => a -> a -> Bool
max :: Ord a => a -> a -> a
min :: Ord a => a -> a -> a
```

The Show Class

```
show :: Show a => a -> String
```

The Num Class and Subclass

```
(+) :: Num a => a -> a -> a
(-) :: Num a => a -> a -> a
(*) :: Num a => a -> a -> a
(/) :: Fractional a => a -> a -> a
div :: Integral a => a -> a -> a
mod :: Integral a => a -> a -> a
(^) :: (Integral b, Num a) => a -> b -> a
sqrt :: Floating a => a -> a
```

Higher Order Function **

```
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
flip :: (a -> b -> c) -> b -> a -> c
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
```

List Function

```
(++) :: [a] -> [a] -> [a]
(!!) :: [a] -> Int -> a
head :: [a] -> a
tail :: [a] -> [a]
reverse :: [a] -> [a]
replicate :: Int -> a -> [a]
zip :: [a] -> [b] -> [(a, b)]
```

The Foldable Class

```
length :: Foldable t => t a -> Int
sum :: (Foldable t, Num a) => t a -> a
elem :: (Foldable t, Eq a) => a -> t a -> Bool
foldl :: Foldable t => (b -> a -> b) -> b -> t a -> b
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
concat :: Foldable t => t [a] -> [a]
concatMap :: Foldable t => (a -> [b]) -> t a -> [b]
```

The Monad Class

```
(>>=) :: Monad m => m a -> (a -> m b) -> m b
return :: Monad m => a -> m a
```

I/O Functions

```
getChar :: IO Char
getLine :: IO String
putChar :: Char -> IO ()
putStr :: String -> IO ()
putStrLn :: String -> IO ()
print :: Show a => a -> IO ()
```

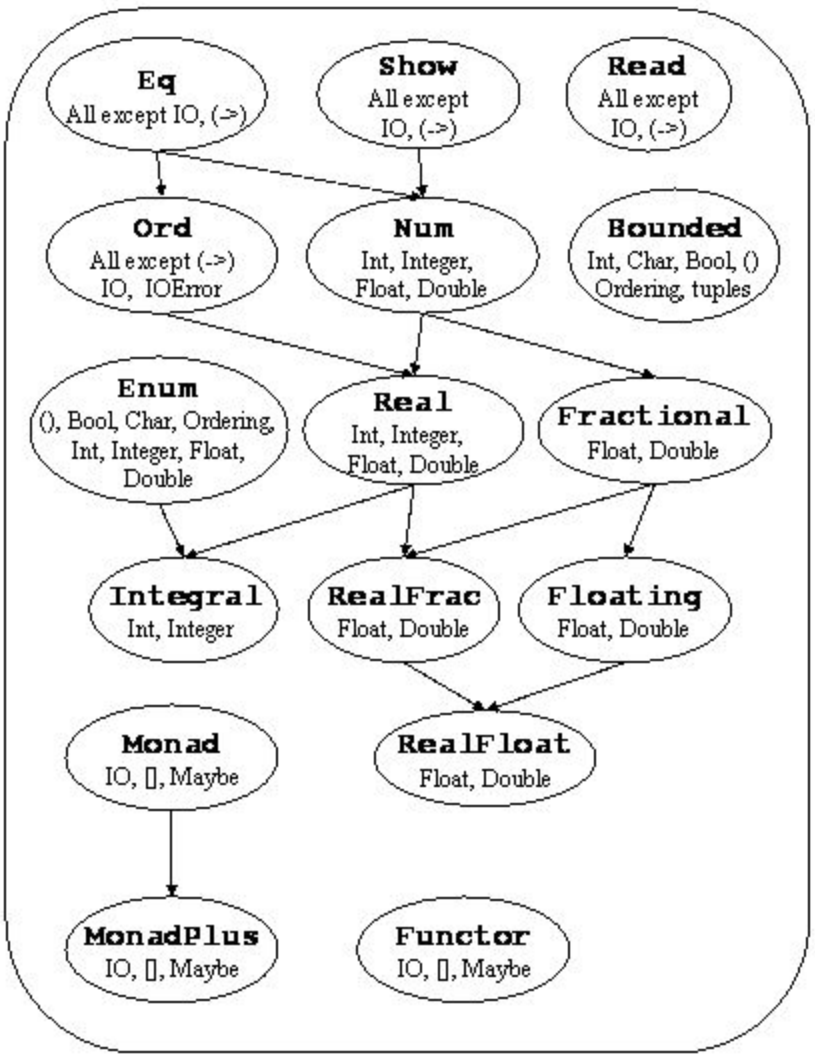
Other Infix

```
(&&) :: Bool -> Bool -> Bool
(||) :: Bool -> Bool -> Bool
(:) :: a -> [a] -> [a]
(.) :: (b -> c) -> (a -> b) -> a -> c
($) :: (a -> b) -> a -> b
```

Other

id :: a -> a
const :: a -> b -> a
not :: Bool -> Bool
otherwise :: Bool
error :: [Char] -> a

Class Hierarchy



Prolog

Control

:Goal1 **,** :Goal2
:Goal1 **;** :Goal2
\+ :Goal
:Condition **->** :Action
:Condition **->** :Action **;** :Else

Predicate

Verify Type of a Term

var(@Term**)**
nonvar(@Term**)**
integer(@Term**)**
float(@Term**)**
atom(@Term**)**
compound(@Term**)**
ground(@Term**)**

Arithmetic

abs(+Expr**)**
between(+Low, +High, ?Value**)**
plus(?Int1, ?Int2, ?Int3**)**

Lists

length(?List, ?Int**)**
member(?Elem, ?List**)**
append(?List1, ?List2, ?List1AndList2**)**
sort(+List, -Sorted**)**
msort(+List, -Sorted**)**
keysort(+List, -Sorted**)**
maplist(:Goal, ?List**)**

Other

setof(+Template, +Goal, -Set**)**
bagof(+Template, :Goal, -Bag**)**
call(:Goal**)**
op(+Precedence, +Type, :Name**)**